

# How Penalty Leads to Improvement: a Measurement Study of Wireless Backoff

Dmitriy Kuptsov<sup>†‡</sup>, Boris Nechaev<sup>†‡</sup>, Andrey Lukyanenko<sup>‡</sup> and Andrei Gurtov<sup>†¶</sup>

**Abstract**—Despite much theoretical work, different modifications of backoff protocols in 802.11 networks lack empirical evidence demonstrating their real-life performance. To fill the gap we have set out to experiment with performance of exponential backoff by varying its backoff factor. Despite the satisfactory results for throughput, we have witnessed poor fairness manifesting in severe capture effect. The design of standard backoff protocol allows already successful nodes to remain successful, giving little chance to those nodes that failed to capture the channel in the beginning. With this at hand, we ask a conceptual question: *Can one improve the performance of wireless backoff by introducing a mechanism of self-penalty, when overly successful nodes are penalized with big contention windows?* Our real-life measurements using commodity hardware demonstrate that in many settings such mechanism not only allows to achieve better throughput, but also assures nearly perfect fairness.

**Index Terms**—Wireless networks, channel access, performance measurement and modeling, protocol design

## I. INTRODUCTION

THERE are several well-known problems associated with the communication in wireless LANs (WLANs). In particular, since nodes in such networks use shared medium in an unlicensed radio spectrum for transmission of frames, collisions are possible. Measurement study of large scale enterprise WLANs [4] showed that nearly 15% of sender-receiver pairs experience significant loss due to collisions. Another problem is fairness. The study in [7] showed that 802.11 networks have good short-term fairness when the number of contending stations is small, e.g. 2, and becomes worse for an increasing number of stations.

In this paper we tackle these two problems by proposing two novel backoff mechanisms for 802.11 networks. The protocols, although different in design, have a similar inspiration and a common goal—to increase throughput and improve fairness. The underlying principle of both schemes is to penalize overly successful nodes with large contention windows and accordingly reward unsuccessful nodes with small contention windows. This principle is inspired by mechanism design [21] from the field of game theory, where the common practice is to measure behavior in terms of Nash equilibrium and variate the equilibrium by controlling the rules. We consider these backoff protocols as the first contribution of the paper.

To evaluate effectiveness of the proposed designs we, unlike many other researchers [12], [18], [29], take a measurement approach. Thus, to conduct the study we implement two existing and two novel backoff protocols on commodity hardware.

We consider the details of implementation, data collection and calibration techniques to be our second contribution.

Next, we conduct a series of real-life experiments in various scenarios. The goal here is to explore how throughput and fairness of different protocols change when we vary such parameters as backoff factor, number of clients and lossiness of the channel. Our third principal contribution is in revealing performance trends and optimal configurations both in real-life experiments and in an analytic model.

To operate properly, the proposed algorithms require an accurate estimation of the number of active wireless stations attached to the access point. Our last contribution in this work is analysis of two novel metrics for estimating the number of active clients. We implement these metrics in a Linux-based wireless access point and evaluate the system performance in an office-like environment.

To preview our results, we have found that in many settings the penalty mechanisms improve fairness greatly. At the same time, such improvement is also accompanied by considerable increase in throughput. To be more specific, our experimental work indicated the following:

- In all the scenarios that did not include hidden terminals, we have witnessed that the backoff with penalty improves throughput by 100% when compared to the standard 802.11 backoff.
- In many settings the penalty mechanisms allowed to achieve nearly perfect fairness. Even with RTS/CTS mechanism disabled, two hidden terminals using backoff with the penalty achieved nearly perfect fairness under certain conditions.
- The penalty mechanisms reduced collision rate on average from 0.3 to 0.14 in lossy environment and from 0.29 to 0.09 in normal environment.
- We have observed that the penalty mechanisms potentially do not increase the delays for delay sensitive UDP traffic when compared to standard backoff protocol.

The rest of the paper is organized as follows. In section Section II we review two existing and introduce two novel backoff protocols that will be studied in this paper. Section III discusses the implementation details, experimental environment, data collection and calibration process. We devote Section IV to our experimental findings and in Section V we outline the theoretical analysis. Section VI describes the protocol for estimating the number of active clients. In Sections VII and Section VIII we discuss the implications of our findings and their correspondence with related work. We make concluding remarks in Section IX.

HIIT<sup>†</sup>  
Aalto University<sup>‡</sup>  
CWC, University of Oulu<sup>¶</sup>

## II. BACKOFF PROTOCOLS

In this section we review the modifications of the backoff protocols we have experimented with.

*Standard backoff with varying backoff factors.* This is the protocol that is currently in use in all 802.11 standards. We use this protocol as a benchmarking baseline and compare it with all other algorithms presented in this paper. The only modification of this protocol that we explore is varying backoff factors  $r \in [1.2, 2.6]$ . We discuss exact changes in Section III. Figure 1 demonstrates how the standard backoff behaves upon succeeding ( $s$ ) or failing ( $f$ ) to send a packet. In case of failure, e.g. due to a collision, the protocol retransmits the packet and increases  $i$  until it reaches  $i = 6$ . If all 7 retries fail, the packet is discarded and similarly to the result of a successful transmission,  $i$  is set to 0.

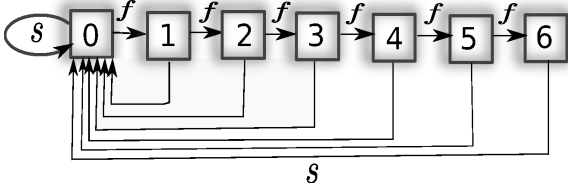


Fig. 1: Standard backoff

*Backoff with penalty.* This is the first novel protocol we propose in this paper. The idea behind it is to penalize the stations that always successfully transmit frames by throwing them to a state with a larger contention window, and at the same time allow the stations that are unsuccessful in transmissions to have smaller contention windows. As seen in Figure 2, this is achieved by putting the node which was in state 0 and had a successful transmission into state 6. Such a modification increases the chances of each station to seize its portion of wireless medium. Similarly to standard backoff, after 7 unsuccessful retries the mechanism retreats to state 0. The form of this modification is in spirit with mechanism design, a field of game theory, where the common practice is to measure behavior of players in terms of Nash-equilibrium and variate the equilibrium by controlling the rules (e.g., by punishing misbehavior) [21]. As an outcome of punishing undesired behavior the system gets the desired properties.

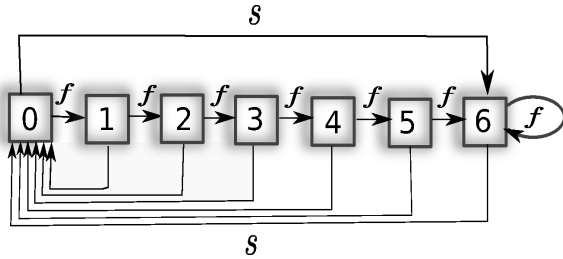


Fig. 2: Backoff with penalty

*Rollback backoff.* In contrast to backoff with penalty, in our second proposed algorithm (rollback backoff) nodes always start with the state that has the largest contention window, and only nodes that are unsuccessful are rewarded with smaller contention windows. The state transition diagram for this

protocol is presented in Figure 3. Unlike the previous two protocols, 7 failed retries set the state to 6.

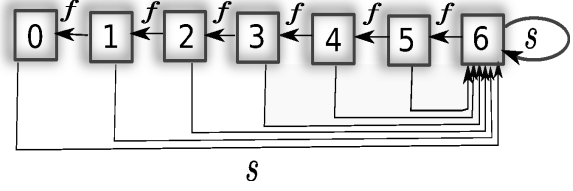


Fig. 3: Rollback backoff

*Backoff with fixed contention window.* This protocol is different from all protocols described above in that it has a single state. In other words, the nodes have up to 7 retries (this is common for all protocols discussed in this paper) but after each failure or success the contention window remains unchanged. Though as we show later, optimal window size should depend on the number of active stations. Contention window sizes that we have used in our experimental work are identical to those presented in Table II.

## III. IMPLEMENTATION AND DATA COLLECTION

In this section we describe the Linux implementation of backoff protocol for 802.11 networks, our experimental testbed and the way the data was collected and calibrated.

### A. Implementation

Most, if not all, hardware manufacturers implement such low level protocols as 802.11 backoff in firmware or in proprietary kernel modules. And typically, the firmware is shipped as a binary microcode that is loaded by OS drivers directly into device memory. This significantly complicates any changes to default protocol behavior. Fortunately, research community and open source enthusiasts provide a great opportunity to explore the internals of such software.

In our implementation we have used open source firmware [1] for Broadcom B43 wireless cards as a basis for experimental work. The code is a result of a meticulous reverse engineering effort done in OpenFWWF project. The microcode is written in assembly language and features implementation of standard 802.11 Medium Access Control (MAC) mechanisms for Broadcom/Airforce chipsets.

While the modifications that we have introduced are minimal, it took us a while to understand various bits and pieces of the firmware. This being said, we found it exciting to understand how the memory is organized and accessed, what are the specifics of particular microcode and how the protocol is implemented in the firmware itself. There are also some other researchers who reported their experience working with the same hardware [28].

Without going into various peculiarities we will instead describe what noticeable changes we have introduced to the firmware. The task was to modify the exponential backoff, as well as random backoff, and the way it is calculated in firmware. In original firmware the contention window is calculated as follows. Initial contention window  $CW_{min}$ , which is also the minimum one, is set to a default value of 31, and

the maximum is bounded with  $CW_{max} = 1023$ . Whenever the collision is detected the client increases the window by a factor of 2. Below we show the snippet of original assembly code that allows to achieve this.

```
1: #define DEFAULT_MAX_CW 0x03FF
2: #define DEFAULT_MIN_CW 0x001F
3: #define DEFAULT_RETRY_LIMIT 0x0007
...
4: orx l4, 1, CUR_CONTENTION_WIN, 0x001, CUR_CONTENTION_WIN;
5: and CUR_CONTENTION_WIN, MAX_CONTENTION_WIN, CUR_CONTENTION_WIN;
```

The contention window is increased by shifting the value in the register `CUR_CONTENTION_WIN` to the right (line 4 in the code above). To ensure that the value of current contention window will not exceed the maximum allowed value 1023, bitwise *AND* operation with the `MAX_CONTENTION_WIN` value is performed. We introduced several changes to the above scheme.

**Non-standard backoff factors.** Instead of using a fixed backoff factor  $r = 2$ , we allow  $r \in \mathbb{R}$  and experiment with the values in  $[1.2, 1.3, \dots, 2.6]$ . In our code  $CW_{min}$  is reduced from 31 to 15, since IEEE 802.11g uses the latter. We also removed  $CW_{max} = 1023$ , which was bounding the growth of  $CW$ . Thus, in our implementation contention window changes according to  $CW_i = CW_{min} * r^i - 1$ , where  $i \in [0, 6]$  is the retransmission counter ( $i = 0$  means that the packet is being sent for the first time). For the experiments with fixed contention window, we have set identical values to all 7 states. For example, for the experiments with 3 clients the contention window for all states was set to 21 [12].

Using floating point factors in original assembly code as above would require the support for floating point multiplication in hardware, which is of course missing. As a workaround we have done the following. First, using trial and error method we have found the offset in memory (note the available shared memory in the devices is just 4KB) which was not used by any part of the code. Second, for every  $r$  we have precomputed possible contention window sizes and corresponding bit masks (we will discuss the need for such bit masks later). The mask was calculated as the minimum value  $2^x - 1$  which is larger than the corresponding contention window. We show the precomputed array for backoff factor  $r = 1.2$  in the snippet of code below.

```
#if 1 //BACKOFF_1_2
/* Start initializing backoff intervals for exponent 1.2 */
mov SHM_BACKOFF_OFFSET, SPR_BASE5;
mov 0x00F, [BACKOFF_0, off5]; /* SHM_BACKOFF_0 = round(16*1.2^0) = 15 */
mov 0x012, [BACKOFF_1, off5]; /* SHM_BACKOFF_1 = round(16*1.2^1) = 18 */
mov 0x016, [BACKOFF_2, off5]; /* SHM_BACKOFF_2 = round(16*1.2^2) = 22 */
mov 0x01B, [BACKOFF_3, off5]; /* SHM_BACKOFF_3 = round(16*1.2^3) = 27 */
mov 0x020, [BACKOFF_4, off5]; /* SHM_BACKOFF_4 = round(16*1.2^4) = 32 */
mov 0x027, [BACKOFF_5, off5]; /* SHM_BACKOFF_5 = round(16*1.2^5) = 39 */
mov 0x02F, [BACKOFF_6, off5]; /* SHM_BACKOFF_6 = round(16*1.2^6) = 47 */

mov SHM_BACKOFF_MASK_OFFSET, SPR_BASE5;
mov 0x00F, [BACKOFF_MASK_0, off5]; /* SHM_MASK_BACKOFF_0 = 16 - 1 */
mov 0x01F, [BACKOFF_MASK_1, off5]; /* SHM_MASK_BACKOFF_1 = 32 - 1 */
mov 0x01F, [BACKOFF_MASK_2, off5]; /* SHM_MASK_BACKOFF_2 = 32 - 1 */
mov 0x01F, [BACKOFF_MASK_3, off5]; /* SHM_MASK_BACKOFF_3 = 32 - 1 */
mov 0x03F, [BACKOFF_MASK_4, off5]; /* SHM_MASK_BACKOFF_4 = 64 - 1 */
mov 0x03F, [BACKOFF_MASK_5, off5]; /* SHM_MASK_BACKOFF_5 = 64 - 1 */
mov 0x03F, [BACKOFF_MASK_6, off5]; /* SHM_MASK_BACKOFF_6 = 64 - 1 */
#endif
#if 0 //BACKOFF_1_3
...
```

Third, the client keeps track of backoff counter variable which is increased by 1 (line 4 below) whenever the collision occurs or resets it to 0 (line 8) whenever the contention window returns to the minimum  $CW_{min}$ . Observe, that the actual minimum contention window doesn't depend on  $r$  and is always equal to 15. Also, for  $r = 2$  our scheme is identical

to the one used in original IEEE 802.11g protocol.

```
1: #define DEFAULT_MAX_CW BACKOFF_6
2: #define DEFAULT_MIN_CW BACKOFF_0
3: #define BACKOFF_LIMIT 0x0006
...
4: add BACKOFF_COUNTER, 0x001, BACKOFF_COUNTER;
/* IF BACKOFF_COUNTER > MAX BACKOFF_COUNTER = CW[BACKOFF_LIMIT] */
5: jle BACKOFF_COUNTER, BACKOFF_LIMIT, dont_adjust_counter;
6: mov BACKOFF_LIMIT, BACKOFF_COUNTER;
7: dont_adjust_counter::
...
8: mov DEFAULT_MIN_CW, BACKOFF_COUNTER;
...
```

Finally, we have changed the way random backoff value from the interval  $[0, CW_i]$  is selected in the firmware. In the original version this was achieved using a bitwise *AND* operation of the pseudo-random number generated in `SPR_TSF_Random` register with the value of current contention window stored in `CUR_CONTENTION_WIN`. This allows to trim the 16-bit random number to the number of bits in the current contention window and thus obtain a random number in  $[0, CW_{min} * 2^i - 1]$ . Though since in our case the values of contention windows are not always powers of 2, the above approach wouldn't work. In principle, it is possible to obtain the required random value in  $[0, CW_i]$  by taking  $(SPR\_TSF\_Random \bmod CW_i)$ . Since division operation is absent in the microcode instruction set, we can subtract  $CW_i$  from `SPR_TSF_Random` until the result is less than  $CW_i$ . We implemented this algorithm, but found that big number of frequent subtraction operations significantly degrades overall performance. Thus, we switched to a more efficient solution. Given a  $CW_i$  we do bitwise *AND* operation between `SPR_TSF_Random` and the precomputed mask equal to the minimum value of  $2^x - 1$  bigger than  $CW_i$ . This gives us a random number  $H \in [0, 2^x - 1]$ . If  $H \leq CW_i$ , then we have found the sought random backoff value. Otherwise, if  $H \in (CW_i, 2^x - 1]$ , we repeat the above operations of taking a new 16-bit random number and applying bitwise *AND* with the mask, until  $H \leq CW_i$ . Empirical evidence shows that the loop doesn't cause any noticeable performance issues.

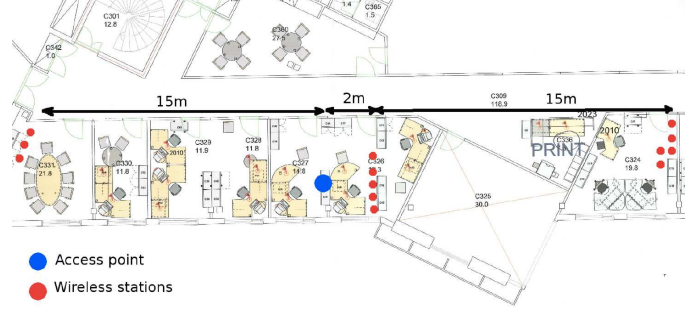
**Non-standard state transition.** To implement rollback backoff and backoff with penalty we have also changed the way the clients select backoff states whenever collision happens. For instance, for the rollback backoff protocol instead of starting from state 0 and increasing the backoff counter by one upon every collision, the clients start from state 6 (maximum contention window) and decrement the counter on each retry.

For backoff with penalty we have also added logical check to a callback which is invoked after successful frame transmission. If the packet was successfully transmitted at the first attempt then the state becomes 6, otherwise the state is set to 0:

```
1: je BACKOFF_COUNTER, DEFAULT_MIN_CW, move_to_last_state_2;
2: or MIN_CONTENTION_WIN, 0x000, CUR_CONTENTION_WIN;
3: mov DEFAULT_MIN_CW, BACKOFF_COUNTER; /* RESET TO 0th STATE */
...
4: move_to_last_state_2::
5: or MAX_CONTENTION_WIN, 0x000, CUR_CONTENTION_WIN;
6: mov DEFAULT_MAX_CW, BACKOFF_COUNTER; /* MOVE TO 6th STATE */
```

## B. Experimental environment

Our testbed comprised only commodity hardware. We have selected inexpensive (\$4 each) 802.11g wireless cards, 4 commodity computers, a 100/10 Base-T switch and single



(a) Idealized environment: nodes are localized in one place

(b) Normal environment: nodes are scattered around the office

Fig. 4: Experimental testbeds

wireless access point running OpenWRT Linux, which also supported 802.11g standard. The variants of our testbed are shown in Figure 4.

We have dedicated a single computer to play the role of a *master node*. This node was responsible for sending commands to slave nodes to start experiments and also was participating in receiving and sending test traffic from and to slave nodes over wireless interfaces. The other 3 machines were used as *slave nodes*. These nodes were provisioned with a single wired connection each and 5, 4 and 3 wireless cards correspondingly.

Since slave nodes had multiple interfaces, we configured these Linux boxes with policy based routing rules. This allowed us to send specific traffic through a specific interface. For instance, all control traffic such as commands and calibration packets (discussed below) was sent to wired interfaces, while wireless cards were performing experimental bulk transfers. Such setup allowed us to avoid interfering control traffic with experimental wireless flows. Since the experiments were conducted in an office-like environment, we have configured the wireless network with a channel that was least used and thus most probably not overlapping with other channels.

### C. Data collection and calibration

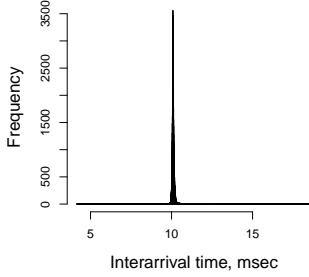
Our experiments consisted of three traffic types: bulky TCP upload and download streams, and delay sensitive UDP flows. We have generated the first traffic pattern with the transfers of 64 MB files using Linux `wget` utility. Here the master node was downloading the files from the slaves. Note that from the perspective of slave nodes (stations) this is the upload traffic. The second traffic pattern was generated using similar file sizes, but now the master node was uploading the file to the slave nodes using `scp` utility (hence we call it download traffic in Section IV-D). We could implement a simple custom application running on top of UDP to get more control over rate limiting and congestion control dynamics. Though we chose to use TCP, since it's the most popular transport protocol for performing large file transfers. We have used UDP in the experiments involving measurement of delays. In particular we have used UDP flows to emulate delay sensitive voice-over-IP sessions. We have implemented a custom application

that was periodically (every 10ms) sending 100 bytes packets from slave nodes to the master node. Using a socket option the master node was registering timestamps of each received UDP frame. This allowed us to achieve the required precision in measurements.

We instructed the kernels on slave machines to log on per packet basis the information about number of retries, acknowledgment flags, packet size, used contention window and backoff interval. In doing so we have encountered a problem with Linux kernel which did not allow us to log this information too frequently. To overcome the issue we have recompiled the kernel with an increased ring buffer size for debug messages and also increased kernel `printk` rate limit. Upon receiving packet transmission status notification from firmware, the kernel registers the event and logs it into ring buffer. The ring buffer is periodically (every 0.1 seconds) read and dumped into a file. Each event was also flagged with the wireless interface ID and a timestamp. For the latter we found it easier to use time elapsed since machine bootup rather than time since the Unix epoch.

Upon inspecting our logs, we discovered that some of them still contained few errors of two different types. First is transmission error reported by the wireless card driver. Unfortunately we failed to identify the cause for the error. Second is the kernel debugging error mentioned above—apparently even after tweaking kernel parameters, it still didn't manage to log all events and occasionally reported that some of them were suppressed. We estimated share of these two errors relative to the number of benign log events and found it to be negligible, well below 1%. Hence, we decided to ignore erroneous log messages by discarding them.

In total we used 12 wireless cards installed on three slave nodes. For majority of experiments we have used 3, 6, 9 and 12 concurrently active clients. We have balanced the usage in such a way that for any number of active clients we have employed all three slave nodes in our testbed. Our eventual goal was to study combined performance of all active clients, which required merging logs recorded on different machines. Since clocks on the machines were not in sync, we had to find a way to correctly align our logs. The solution was to send calibrating beacons from the master node to all slave



**Fig. 5:** *Interarrival time between consecutive beacons.*

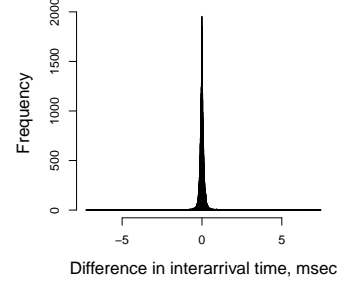
machines via wired interfaces. In principle, it would have sufficed to send a single beacon in the beginning of each experiment, which slave nodes would have recorded as a reference time frame. Then subtracting this value from each packet's timestamp would yield a relative packet's timestamp in the merged log file. Alas, this solution is not perfect, since in prolonged experiments clock drift among different machines could cripple relative packet timings by putting some of them unduly further into the future or the past. To eliminate the effect of clock drift we instructed the master node to send beacons periodically, with an interval of 10 msec. This made it possible to do re-alignment on short timescales.

Another task where we utilized beacons is splitting logs into bins. Binning allowed us to do fine-grained analysis of various performance metrics and trace evolution of different relevant parameters over time. We found it convenient to perform binning by tying bin size to a number of consecutive beacons. For instance, 10 beacons would correspond to a 100 msec bin, 100 beacons - to a 1 sec bin, etc.

Even though the beacons were sent with a strict 10 msec interval, there is no guarantee that they were recorded by slave nodes with exactly the same intervals. Perfect interarrival time between beacons could be skewed by various network, NIC or OS effects. Incorrect beacon interarrival times could result in imperfect binning and thus undermine any analysis that relies on the assumption of constant bin size. To assess the possible skew in beacon timestamps, we calculated beacon interarrival times for all logs. A typical distribution is shown in Figure 5. To our relief, interarrival times turned out to be sharply clustered around 10 msec. Though the figure still shows rare outliers. This could lead to drift in cumulative beacon intervals among several machines. However, Figure 6 eliminates this concern. To make the plot we calculated differences between respective beacon interarrival times on different machines. The facts that the distribution is centered at zero, highly clustered and symmetric proves that bins calculated based on beacons are staying equally sized in the long run.

After devising start and end timestamps of each bin, we traverse our logs and using all packets falling into it calculate number of successfully transmitted packets, number of failures and the total number of packets in the bin. The set of these three parameters is also calculated for each client participating in the experiment.

The last bit of calibration that we performed before moving



**Fig. 6:** *Difference in beacons interarrival times for different machines.*

to data analysis is truncating our logs to all-active periods. Packet sending may have started at different times on different clients. Also, some clients may have finished sending faster than the others. In our analysis we were interested in those periods when all clients were active, since this would give us confidence that all of them were involved in competing for the channel. To identify such period, we calculated the maximum timestamp of the first full-size (1540 bytes) packet among all clients and the minimum timestamp of the last one. Then, using these two values we discarded all packets before and after them respectively. We made sure that the remaining periods were big enough to provide meaningful data for our analysis.

#### IV. EXPERIMENTAL RESULTS

Our research agenda is to observe behavior of all protocols described in Section II in various environments. The goal is to reveal meaningful trends by experimenting with different settings and setups. In this section we present the results for throughput (which is measured as an aggregated throughput of all stations), fairness, collision probability and delays obtained for various values of backoff factor as well as with MAC layer rate control enabled and disabled. We begin with the environment in which nodes are close to access point ( $< 1m$ , Figure 4(a)). Although we call this environment idealized, we found that even in such simple setting we had multipath reflections from the metal rack, which brings the results obtained in this environment closer to real life. Next, we present results for the settings where the nodes were deployed in the environment which is typical to many office-like deployments of 802.11 networks. This experiment is followed by the scenario with two hidden stations. In all three above mentioned experiments the nodes were uploading a large file to the server. We conclude our empirical evaluation with experiments that involve two additional traffic patterns. The first pattern we consider is when all nodes were downloading a large file. Here our intention was to observe whether the behavior of the protocols will change in comparison to the experiment with upload traffic. For the second pattern we have emulated a mixed traffic pattern with the presence of low rate voice-over-IP (VoIP) like flow and multiple (upload) TCP streams. The idea was to observe the impact of bulky streams on less demanding, but delay sensitive, flows. In this experiment we were mainly interested in delay characteristics.

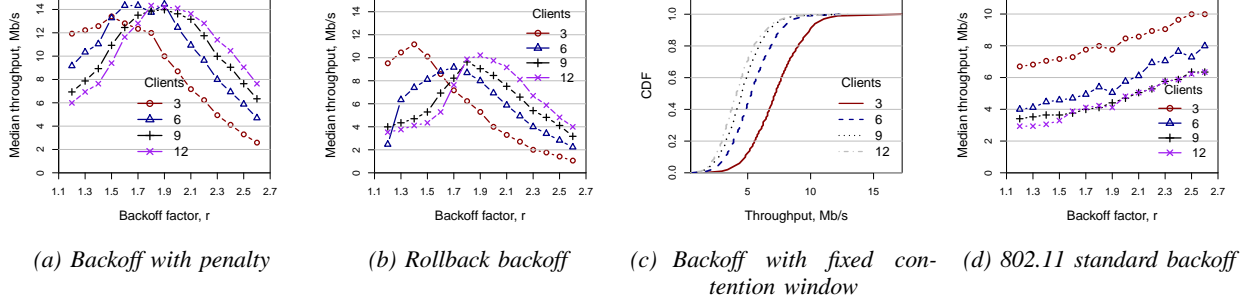


Fig. 7: Throughput. Idealized environment. MAC layer rate control enabled

TABLE I: Median throughput (Mb/s) comparison. Idealized environment

Protocol Clients	Penalty	Rollback	Std, r=1.2	Std, r=2.0	Std, r=2.6	Fixed CW
3	12.57(r=1.4)	11.16(r=1.4)	6.69	8.45	9.98	7.16
6	14.33(r=1.7)	9.16 (r=1.7)	3.99	5.75	7.98	5.28
9	13.86(r=1.8)	9.63 (r=1.8)	3.4	4.69	6.34	4.46
12	14.09(r=2.0)	10.22(r=1.9)	2.9	4.8	6.34	4.11

But before we dive into the discussion of the results, we define a metric we have used to measure fairness throughout the paper. Intuitively, fairness is the ability of all transmitting stations to share the channel bandwidth roughly equally in a given period of time. To formalize the above notion, we have used *Jain's fairness index* [13] with a sliding normalized window of varying sizes. In other words, if  $N$  is the number of stations generating traffic, we have computed Jain's index for windows  $w \in \{N, 2N, 3N, \dots, xN\}$ . Here window size is expressed in number of packets and the lower it is the shorter term fairness is considered. More formally, if  $\tau_j$  is the fraction of successful transmissions for station  $j$  in window  $w$ , then the fairness index is defined as:

$$F(w) = \frac{(\sum_{j=1}^N \tau_j)^2}{N \sum_{j=1}^N \tau_j^2} = \frac{1}{N \sum_{j=1}^N \tau_j^2} \quad (1)$$

Thus  $F(w) = 1$  is an indication of perfect fairness, and  $F(w) = 1/N$  is an indication of total unfairness.

#### A. Idealized environment

We begin with the experiment in which nodes are placed close to access point ( $< 1m$ , Figure 4(a)). In the proceeding paragraphs we discuss the results for the following performance metrics: throughput, fairness and collision probability.

The results for the experiments when rate control was enabled are presented in Figure 7. For each backoff factor we calculated median throughput across all throughput values for each bin. The exception is the plot with fixed contention window (Figure 7(c)) in which we plot the whole CDFs. We have also calculated mean throughput values only to find that they are close to median values shown in the plots, thus presenting them here is redundant.

Furthermore, in Table I we highlight the above results using median throughput for most interesting configurations.

Note that for penalty and rollback backoff protocols we use such values of  $r$  which correspond to the maximum values of throughput. In other words, we use the peak values of  $r$  according to figures. Accordingly, we make the following observations which we believe are interesting:

*Observation 1.* For small values of  $r$  (e.g. 1.2) standard 802.11 backoff protocol does not show good throughput. Fairness for such a small backoff factor is comparable to that observed for standard backoff with  $r = 2.0$  (Figure 8(d)).

*Observation 2.* For large values of  $r$  (e.g. 2.6) standard 802.11 backoff shows considerable improvement in throughput. However, this is an illusion, since such improvement is accompanied by drastically bad fairness. As clearly seen in Figure 8(d), for  $r = 2.6$  fairness is worse than for  $r = 2.0$ . In other words, such a big backoff factor allows few hosts to capture the channel, while other hosts are forced to remain silent. If we put this in terms of user experience, it would mean that only few users will complete the download, while for rest of the users the TCP connection will most likely time out (indeed, we have observed such behavior during our experiments). For this reason in the rest of the paper we focus only on  $r = 2.0$  for standard backoff protocol when comparing it with other protocols.

*Observation 3.* For standard 802.11 backoff and backoff with fixed contention windows throughput degrades as the number of clients increases. In contrast, Table I shows that optimal throughput in backoff with penalty and rollback backoff protocols (for  $N \in [3, 12]$ ) is more or less constant, i.e. doesn't change considerably with  $N$ .

*Observation 4.* From the values presented in Table I we have calculated the average improvement of rollback backoff over standard backoff (77%) and over backoff with fixed contention windows (96%). Similarly, the improvement for backoff with penalty is 145% and 172% correspondingly. Interestingly, such improvement in throughput is by no means penalized by lack



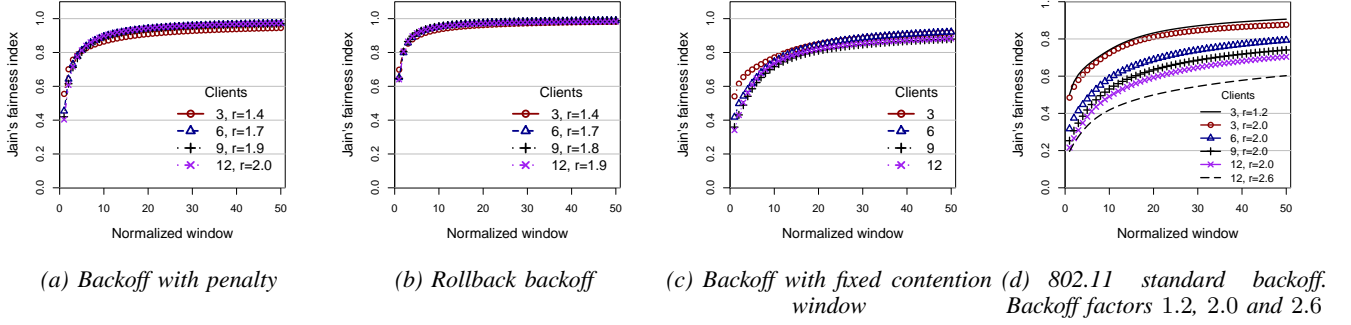


Fig. 8: Fairness. Idealized environment. MAC layer rate control enabled

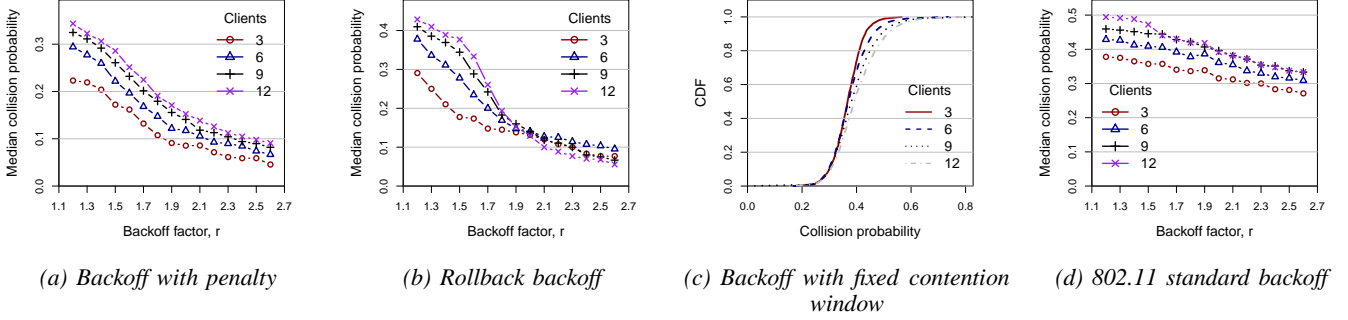


Fig. 9: Collisions. Idealized environment. MAC layer rate control enabled

of fairness as shown in the next paragraphs.

We would like to stress it once again since we believe that this is the most important observation: the increase in throughput for backoff with penalty and rollback backoff does not harm fairness. Moreover, it can be clearly seen in Figures 8(a) and 8(b) that both these protocols achieve nearly perfect short term fairness. On the other hand, unlike other simulation and measurement studies that claim that backoff with fixed contention windows reaches perfect fairness [7], [10], [12], we could not make the same conclusion. As it was expected, the standard 802.11 backoff protocol shows even worse results (Figure 8(d)).

Both backoff with penalty and rollback backoff protocols (for optimal values of  $r$ ) have sufficiently smaller fraction of collisions compared to backoff with fixed contention window and standard protocol. As can be deduced from Figure 9 median collision rate in optimal throughput points varies between 0.14 and 0.2 for backoff with penalty and between 0.15 and 0.21 for rollback backoff. The similar ranges in Figures 9(c) and 9(d) are considerably larger: from 0.37 – 0.4 and 0.3 – 0.4 respectively. Lower collision rate introduces two positive implications. First, it has a direct connection to increase in the throughput; second, it improves the energy efficiency of the protocols as stations spend less time in transmitting the same amount of information.

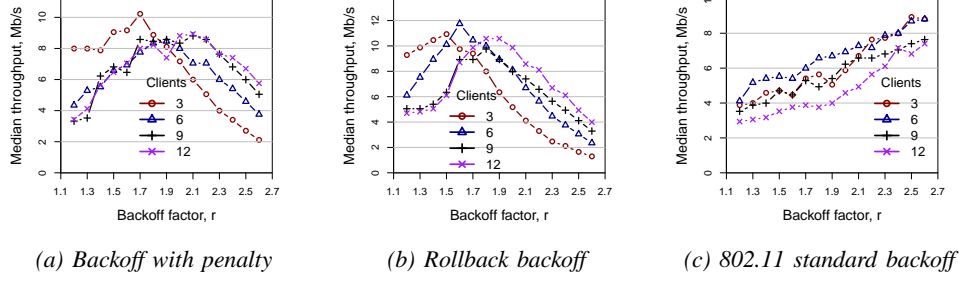
As a side note, we mention that we have repeated the experiments with MAC layer rate control disabled. We observed that for all protocols the median throughput was close to 15Mb/s.

This value for standard 802.11 backoff mechanism was the same for all values of  $r$ , while for rollback backoff and backoff with penalty the shapes of the curves were identical to those as in experiments with MAC layer rate control enabled (meaning that there were peaks that corresponded to highest throughput). However, fairness for all protocols remained similar to that when MAC layer rate control was enabled.

We think that the results obtained when MAC layer rate control was enabled have greater importance than the identical results with rate control turned off. There is extensive evidence, both theoretical and empirical, that different coding schemes deliver different bit error rates (BER) under the same signal-to-noise (SNR) level. For example, packets transmitted at 1Mb/s can sustain higher noise than packets transmitted at 54Mb/s. In real networks SNR changes dynamically which increases the role of rate adaptation mechanism [19], [31].

### B. Normal environment

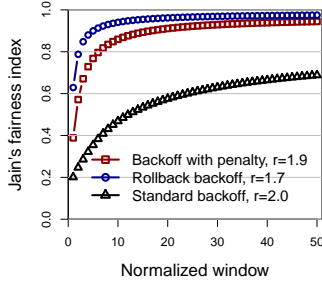
The deployment we have dealt with so far in our experimental work considers the nodes scattered around the access point no further than one meter. Although this setting allowed us to test the utility of the proposed backoff protocols, to affirm ourselves and the reader we have conducted a set of additional experiments where nodes are placed away from each other by more than 30 meters. Such deployments appear in many home and office environments. Therefore, the results we obtain in this setting allow us to judge the applicability of the penalty-based protocols in realistic settings. Note, in this experiment



**Fig. 10:** Throughput. Normal environment.  $> 30m$  distance between nodes

we had 4 other operational wireless networks, and several networks with partially overlapping channels. Schematically the deployment looks as shown in Figure 4(b)

There is no doubt that trends repeat. This is best demonstrated by Figure 10. Similarly to the experiments in idealized environment described in previous section we have calculated the average improvement of the penalty based protocols over the standard 802.11 backoff protocol. The improvement for the rollback backoff was  $> 70\%$  in comparison to standard backoff. The result for the backoff with penalty was more modest  $> 55\%$ . Finally, fairness of backoff with penalty and rollback backoff was again much better than in the other two protocols (Figure 11). We show the fairness results for 12 stations only because they are almost identical to those obtained in the idealized environment.



**Fig. 11:** Normal environment. Fairness. 12 stations

### C. Hidden stations

A hidden terminal is a serious problem for wireless networks. Briefly, it can be described as a scenario when two stations are placed far enough (or have an obstacle in between) so that they can't detect electromagnetic radiation from each other. Clearly, this breaks down the operation of DCF protocol and as such is a highly unwanted network condition. And it is exactly the reason why and how we have designed our next experiment. Regarding the goal of the experiment, here we wanted to observe whether the penalty and reward mechanisms, built-in in our backoff protocols, can solve the problem without additional mechanisms such as RTS/CTS.

To model such configuration we have placed two stations so that they could detect signals from the access point only

but not from each other. To confirm this, we have configured one hidden station in the access point mode and tried to detect beacons from it on the other hidden terminal. We repeated the test in reverse direction to ensure that the property holds for the second hidden terminal. Another desirable property of this setting was the difference in signal strengths produced by the two stations. Thus, it was our intention to have  $\sim 10-15dBm$  difference in signal strengths for two stations connected to the access point. To achieve this we have done the following: first, we have placed one station inside a metal rack (see Figure 4(a)); and second, we have wrapped antennas of both stations with metal foil.

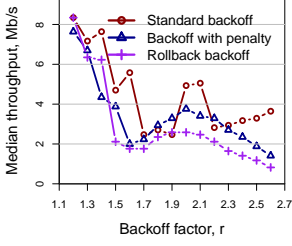
Figure 13 illustrates fairness with Jain's index  $w = 100$ . There is little to no fairness for standard backoff protocol, which happens due to a single station capturing the whole channel, irrespectively of  $r$ . There is almost no fairness for rollback backoff with  $r \leq 1.4$ , after which it starts growing for higher values of  $r$ , and eventually reaches perfect fairness. For backoff with penalty the picture is similar. It is hard to simultaneously achieve good throughput and fairness in this scenario. We further discuss this issue in Section VII. However, both modifications of the protocol show similar throughput results as the standard protocol, which is good.

### D. Download traffic pattern

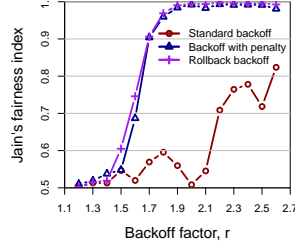
In most wireless networks (home and public WI-FI, non-mesh networks) volume of download traffic is typically higher than volume of upload traffic. For this reason, in this section we investigate how the backoff protocols behave in a situation when download traffic pattern prevails. In this experiment the access point was following standard backoff protocol and stations were running both modified and non-modified versions of the protocols. Our intention was to understand the effect of different backoff protocols (deployed on the stations) on TCP performance. In contrast to upload scenario discussed in previous sections, in this case data packets are sent by the access point to the stations. Thus backoff protocols implemented on the stations manifest themselves in control over sending TCP acknowledgment packets. These packets are typically much smaller than data packets and due to TCP's "ACK every second" policy are approximately twice as rare.

We present the results for throughput in Figure 14. Backoff

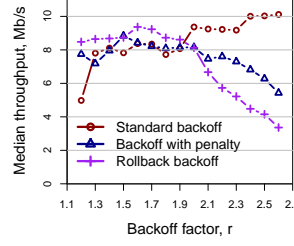




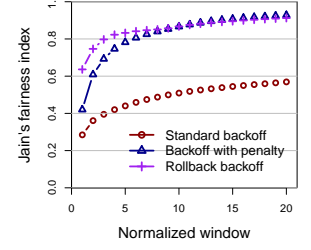
**Fig. 12:** Experiment with 2 hidden stations. Throughput



**Fig. 13:** Experiment with 2 hidden stations. Fairness



**Fig. 14:** Experiment with down-load traffic. Throughput



**Fig. 15:** Experiment with down-load traffic. Fairness

with penalty and reverse backoff do show the highest throughput for optimal  $r$  (in this experiment  $N = 7$ , therefore optimal  $r$ , according to Table II, is 1.5 for rollback backoff and 1.7 for backoff with penalty correspondingly). Although, these results are comparable to those observed for the standard backoff ( $r = 2.0$ ). To our surprise, even in such configuration backoff with penalty and rollback backoff, according to Figure 15, both achieve far superior fairness than the standard protocol.

This experiment clearly demonstrates the benefit of the two modified protocols for the networks with prevailing download traffic.

#### E. Delay sensitive traffic

Our final experiment was designed to understand the impact of our protocols on delay sensitive traffic under the presence of bulky TCP flows. We started concurrent TCP streams on 9 clients and at the same time a single low rate UDP flow on another client. In this setting the UDP flow, limited to  $80Kb/s$ , was emulating a VoIP-like call. Such rate is roughly equivalent to sending 100 byte packets every 10ms.

To generate UDP traffic we have implemented a simple client-server application in C language. The client was periodically sending UDP packets and the server timestamped the received packets using the socket option. While there were several sources of noise in the measurement, including clock skew on the server and imperfections of *usleep(usec)* function, these were rather insignificant so we ignored them (we have measured these imperfections with synthetic tests to observe the error).

As we have stated earlier, the goal of the experiment was to understand how mixed traffic impacts delays in less demanding flows. We present these results in Figure 16. The most crucial observation here is that for  $r_{opt}$  when  $N = 9$  (see Table II) the mean delays of penalty and rollback protocols are low and close to 10ms. This can be explained by the reduced number of packet collisions, which helps avoid MAC layer retransmissions. On the other extreme, in Figure 16 one may notice the delay increases for  $r > 2.0$ . We think this is due to the backoff intervals being undesirably large which increases the overall waiting of the queued packets. However, this situation will be similar for all backoff protocols.

#### V. ANALYTICAL MODEL

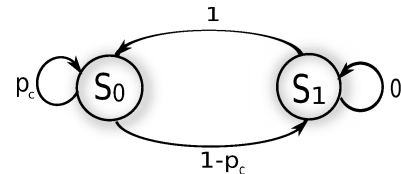
In this section we derive an analytical model describing behavior of wireless networks with rollback and penalty backoff protocols. We consider the model important for two reasons. First is to solidify and corroborate our empirical results—as we will see later in this section the proposed model agrees well with our experimental findings. Second, the model can be used to choose an optimal value of  $r$  for a given number of active stations  $N$ , importance of which is discussed in Section VII.

*Relationship between expected contention window and  $r$ .* Using many lines of research [3], [27], [29] on analysis of 802.11 networks as a starting point we find the expected value of contention window for rollback backoff protocol:

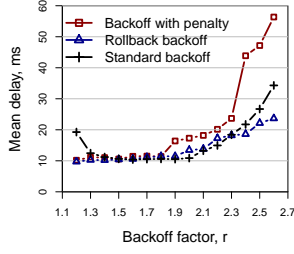
$$E[CW] = \frac{(CW_{min} - 1)(1 - p_c)(p_c^k - r^k)}{2(1 - p_c^k)(p_c - r)}, \quad (2)$$

where  $r$  is the backoff factor,  $p_c$  is the probability of collision (to be defined in Equation 7),  $k = 7$  is the maximum number of retransmissions and  $CW_{min} = 16$  is the minimum contention window.

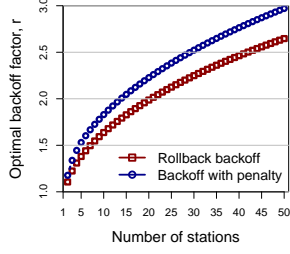
To model backoff with penalty we used an observation that according to Figure 2 the station, besides backoff process itself, can be in two states. The first state ( $S_0$ ) is the state in which the station fails with probability  $p_c$  to transmit at the first attempt and thus follows the standard backoff protocol. The second state ( $S_1$ ) corresponds to the case when the station succeeds to transmit a frame at the first attempt with probability  $1 - p_c$  and is thus forced to use the largest contention window for transmission of the next frame. We can model this process with a two-state Markov chain (Figure 20), which essentially encodes the states the backoff with penalty protocol can be in when attempting to transmit a new frame (not to retransmit a failed frame). The stationary probability distribution of the chain is  $\{1/(2 - p_c), (1 - p_c)/(2 - p_c)\}$ . We use this fact to further define Equation 3 which represents the expected contention window for backoff with penalty.



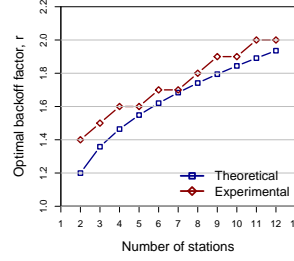
**Fig. 20:** Markov chain for backoff with penalty



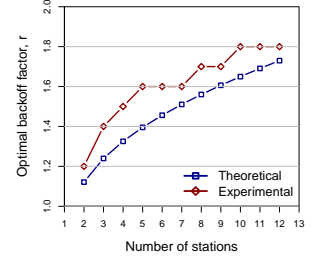
**Fig. 16:** Per packet delays observed for the experiment with mixed TCP and UDP traffic



**Fig. 17:** Comparison of theoretical results for backoff with penalty and rollback backoff



**Fig. 18:** Empirical vs. theoretical results for backoff with penalty protocol



**Fig. 19:** Empirical vs. theoretical results for rollback backoff protocol

$$E[CW] = \frac{1}{2 - p_c} \sum_{i=0}^{k-1} \frac{(1 - p_c)(CW_{min} - 1)p_c^i r^i}{2(1 - p_c^k)} + \frac{1 - p_c}{2 - p_c} \sum_{i=0}^{k-1} \frac{(1 - p_c)(CW_{min} - 1)p_c^i r^{k-1}}{2(1 - p_c^k)} \quad (3)$$

$$= \frac{1}{2 - p_c} \frac{(1 - p_c)}{(1 - p_c^k)} \frac{(CW_{min} - 1)}{2} \left( \frac{(p_c^k r^k - 1)}{(p_c r - 1)} - r^{k-1}(p_c^k - 1) \right)$$

The optimal expected contention windows. By defining the probability of transmission attempt  $p_t$  as:

$$p_t = 2/(E[CW] + 1) \quad (4)$$

similarly to [6], [12] we can further define other probabilities:  $p_n$ ,  $p_s$ ,  $p_c$ . The probability that none of  $N$  competing hosts will send a packet  $p_n$  is:

$$p_n = (1 - p_t)^N \quad (5)$$

The probability  $p_s$  that the host will successfully transmit a packet depends on the probability that the host will attempt to send a packet and none of other  $N - 1$  hosts will do so. It is easy to see that such probability can be defined as follows:

$$p_s = \binom{N}{1} p_t (1 - p_t)^{(N-1)} \quad (6)$$

Lastly, the probability  $p_c$  that at least two hosts will send a packet simultaneously, i.e. collision will occur, can be found by subtracting both  $p_s$  and  $p_n$  from the total probability:

$$p_c = 1 - p_s - p_n = \sum_{i=2}^N \binom{N}{i} p_t^i (1 - p_t)^{(N-i)} \quad (7)$$

With these at hand, throughput can be defined as a function of probability  $p_t$ :

$$F(p_t) = \frac{E[Bytes]}{E[Time]} = \frac{Sp_s}{p_s t_s + p_c t_c + p_n t_n} \quad (8)$$

The values  $t_s$ ,  $t_c$  and  $t_n$  respectively represent: The time to send a packet of length  $S$

$$t_s = DIFS + t_{bo} + TX(K) + SIFS + RX(ACK)$$

The time to detect a collision

$$t_c = DIFS + t_{bo} + TX(K) + SIFS$$

**TABLE II:** Optimal values of  $r$  for backoff with penalty and rollback backoff protocols

N	$CW_{fixed}^1$	$E[CW]$	With penalty, $r_{opt}$	Rollback, $r_{opt}$
2	12.4	14.9	1.18	1.11
3	21.3	27.3	1.35	1.25
4	30.1	40.1	1.45	1.31
5	38.9	55.2	1.53	1.38
6	47.6	71.2	1.65	1.45
7	56.4	88.8	1.67	1.5
8	65.1	107.8	1.73	1.55
9	73.8	128.5	1.78	1.65
10	82.87	150.8	1.85	1.67
11	91.2	174.8	1.88	1.69
12	100	200.5	1.95	1.75

The time during which the network is idle

$$t_n = \text{Slot duration}$$

And finally we define an average backoff time the node waits before transmitting a frame (we have found it surprising that in previous studies this important component was omitted):

$$t_{bo} = \frac{E[CW]}{2} t_n$$

The function  $F(p_t)$  is strictly concave in the interval  $p_t \in [0, 1]$ , and thus by finding its first derivative and equating it to zero we can obtain the value of  $E[CW]$  which yields the maximum throughput:

$$\frac{Np_t - 1}{(1 - p_t)^N} = \frac{t_n - t_c}{t_c} \quad (9)$$

**Numerical results.** To calculate the theoretical results we use  $t_s$ ,  $t_c$  and  $t_n$  which correspond to 802.11g standard working at its maximum rate:  $t_s = 3.22 \cdot 10^{-4} s$ ,  $t_c = 2.92 \cdot 10^{-4} s$  and  $t_n = 9 \cdot 10^{-6} s$ . To compute  $t_s$  and  $t_c$  we have used packet size  $S = 1540$  bytes. We have used Newton's method to solve the Equation 9 numerically for several values of  $N$ . Next, we have used the obtained values of  $E[CW]$  and the similar method to find the roots of Equation 2 and Equation 3. The computed

<sup>1</sup>We have omitted  $t_{bo}$  from the calculations, which resulted in values that are different from  $E[CW]$

results for the first 12  $N$  are shown in Table II and for the first 50  $N$  in Figure 17. These results represent the optimal values of  $r$  for the above values of  $t_s, t_c, t_n$  and  $S$ .

Next, we compare the empirical results obtained in experiments with the model seeded with parameters of packet size, average transmission time and time to detect a collision extracted from the empirical results. To obtain an adequate comparison of the empirical results with the model one needs to have a good estimate of the mean packet size and the time to detect a collision  $t_c$ . Observe that  $t_c$  itself depends on both the packet size and the rate used to transmit the packet. First, we have used our data sets to compute the mean packet sizes. It appeared that for all experiments involving different  $N$  and  $r$  this mean was close to 1000 bytes. Second, across all the data we have calculated the average rate, and using 1000 bytes as the packets size we have found that the average  $t_c$  was close to  $366\mu s$  for backoff with penalty and  $345\mu s$  for rollback backoff. To be more precise, here  $t_c$  was actually computed without  $t_{bo}$  because  $t_{bo}$  depends on  $E[CW]$  and is included in the model already. And finally, using these values as parameters for the model, we have computed the numerical results and compared those against the experimental data (for optimal  $r$ ). The results of the comparison can be seen in Figure 18 and Figure 19. The empirical curves strongly resemble the theoretical curves, especially for the case of backoff with penalty.

## VI. BACKOFF FACTOR SELECTION PROTOCOL

In this section we present the design, implementation and evaluation of the protocol which allows access points to estimate the number of active wireless stations to adapt the backoff factors accordingly in a dynamic way.

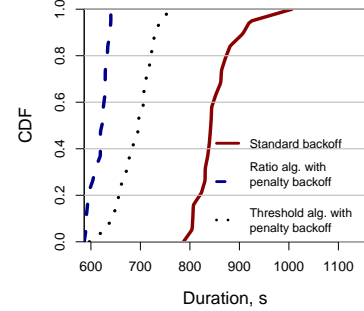
*Algorithms.* We have chosen two different ways to calculate the number of active wireless stations within the access point's vicinity. The first approach is a simple *threshold-based* adaption algorithm. In this algorithm access point counts as active only those associated stations that were transmitting at least  $\tau$  units of time within the update interval  $T$ .

$$N_{active} = \sum_{\forall i} I(\tau_i \geq \tau)$$

where  $I(\cdot)$  is an indicator function mapping logical expression to 1 if it is true, and 0 otherwise. This algorithm is really easy to implement yet it has its drawbacks: possible errors in estimation of currently active number of stations  $N$  if selected  $\tau$  is too small or too large.

Second algorithm that we consider is *fair share or ratio-based* estimation of number of active stations. We show the intuition behind this algorithm with a demonstrative example. Let  $N_{assoc}$  be the number of wireless stations associated with an access point (not necessary transmitting anything). Now let few stations out of these  $N_{assoc}$  be performing bulky transfer, while others be streaming packets with relatively low data rate. Stations with channel occupancy time greater than fair share channel time (bulky high-rate flows) are obviously considered active. For remaining (low-rate) stations we compute their aggregated channel use time and divide it by fair share channel time to estimate additional number of active stations. The

intuition behind such calculation is that occupancy time of multiple low-rate stations can be aggregated, thus treating them as a single station.



**Fig. 21:** Comparison of experiment durations for standard backoff with penalty backoff for which backoff factors adapted using threshold- and ratio-based algorithms correspondingly

Following our example described above we let  $\tau_i$  be the fraction of time, during which station  $i$  was occupying the channel within some update interval  $T$  and define  $x = \frac{\sum \tau_i}{N_{assoc}}$  to be the fair share channel use time for all associated stations. We then formally define the number of active stations as follows

$$N_{active} = \sum_{\forall i} I(\tau_i \geq x) + \left\lfloor \frac{\sum_{\forall j} \tau_j I(\tau_j < x)}{x} \right\rfloor$$

where  $I(\cdot)$  is an indicator function. Because this algorithm can be prone to frequent oscillations in estimates of  $N$  (e.g. if  $x$  is taken exactly equal to fair share), we introduce a coefficient  $\epsilon \in (0, 1]$  and instead of  $x$  we use  $\epsilon \cdot x$  (we use  $\epsilon = 0.8$  in our experiments). This allows the access point not to update the stations with new  $N$  if the difference between between channel use is marginal. We also instruct the access point not to update the stations with new  $N_{active}$  if it differs from previous estimation by at most one, i.e. if  $abs(N_{active}^{new} - N_{active}^{old}) = 1$

Because we strive to make the algorithms described above as much practical as possible we estimate  $\tau_i$  on the access point. First, this approach reduces the overhead due to minimalistic control traffic. Second, in our thinking it is a less error prone method. Thus, to calculate  $\tau_i$  for each station, access point for every packet received from host  $i$  stores the size of the frame ( $P_{i,k}$ ) and the bitrate ( $R_{i,k}$ ) used to transmit  $k$ th frame. And when time  $T$  elapses for each wireless station access point recalculates  $\tau_i = \sum_{\forall k} P_{i,k} / R_{i,k}$ .

*Implementation.* To evaluate the algorithms, we have implemented both algorithms in *hostapd damon* - a piece of user-space software available in Linux that implements wireless access point functionality. We have also introduced a new 802.11 management frame which the access point broadcasts every time it finds new value for the number of active stations. In turn, wireless clients use the value ( $N$ ) conveyed in this frame to select proper current backoff factor. Finally, we have also modified the b43 Linux kernel driver for Broadcom cards which we used in our experimental work (described in

TABLE III: Comparative analysis of backoff protocols

Property \ Protocol	Standard	Fixed CW	Rollback	With penalty
Short-term fairness	<i>Little</i>	<i>Not perfect</i>	<i>Nearly perfect</i>	<i>Nearly perfect</i>
Throughput	<i>Medium</i>	<i>Medium</i>	<i>High</i>	<i>High</i>
Hidden stations (fairness)	<i>Little</i>	-	good for $r \geq 1.6$	good for $r \geq 1.6$
Energy efficiency/Collisions	<i>Low/High</i>	<i>Low/High</i>	<i>High/Low</i>	<i>High/Low</i>
N-adaptable	<i>No</i>	<i>Yes</i>	<i>Yes</i>	<i>Yes</i>

previous sections) to enable dynamical adaptation of backoff factors on wireless cards.

*Evaluation.* With this last experiment we evaluate the performance of backoff factor selection algorithms that we discussed in this section. For this experiment we use the deployment in which wireless stations were scattered around the office (see Figure 4(b)). The idea of the experiment is to vary the number of stations transmitting simultaneously and let the access point specify to wireless stations which backoff factor they should use. We use all 12 stations. Thus, we instruct 6 stations to transmit a 40KB UDP stream every half a minute during 30 seconds interval. The other 6 stations, transmit a 64MB file using TCP protocol. We repeat the experiment 20 times for each of the following settings: (i) all stations and access point were configured with standard backoff protocol, (ii) all stations configured with penalty backoff and access point selects backoff factor using threshold-based algorithm, and (iii) all stations configured with penalty backoff and access point selects backoff factors using ratio-based algorithm. We show the distribution of experiment durations for all settings in Figure 21. Clearly, the *ratio-based algorithm* shows better performance than *threshold-based* algorithm and far better than standard backoff protocol which is expected.

## VII. DISCUSSION

In this section we summarize the key observations we have made during our experimental work. These observations are highlighted in Table III and explained in the proceeding paragraphs.

*Does the penalty mechanism work: short-term fairness as an indicator?* It is a well-known fact that 802.11 standard backoff protocol lacks short-term fairness: nodes in a disadvantageous position fail to compete for the channel as they advance their contention window and hence decrease own odds for occupying the shared medium. Using this fact as a starting point we have introduced two modifications (which have game theoretic roots) to the standard backoff protocol: a penalty and incentive mechanisms. Two have a common goal – improve fairness – but tackle it in different ways. In the first one, successful nodes are penalized with larger contention windows to give a possibility for unsuccessful nodes to compete for the channel. Inversely, in the second protocol, unsuccessful nodes are rewarded with smaller contention windows. Our experiments demonstrate that these two changes can improve fairness of 802.11 networks greatly. On the other hand, unlike other studies [10], [12], our empirical evidence indicates that simply using backoff with fixed contention windows does not guarantee good short-term fairness.

*Increase in throughput, but why?* In many settings we have tried, standard 802.11 protocol does achieve decent throughput. However, such performance is stipulated by capturing the channel by a small fraction of stations. In other words, to some extent, the increase in throughput (when 802.11 standard backoff is used) is a result of decrease in fairness.

On the other hand, when backoff factor  $r$  is selected properly, backoff with penalty shows amazing results due to drastically decreased collisions. In particular, even in lossy environment the average improvement reached almost 100% in comparison to standard 802.11 protocol. These results are closely followed by the results of rollback backoff protocol. And unlike standard backoff protocol, such improvement was not penalized with lack of fairness.

*How will the changes influence energy consumption?* The literature devoted to the problems of energy efficiency in wireless networks is so extensive that we outline few out of many techniques. First, one way to reduce energy consumption by wireless devices is to accurately schedule data transmission and sleeping intervals [30]. The other body of work discusses techniques that allow to reduce the overall amount of packets, e.g. using compression and packet aggregation [2], [22].

Although not specifically designed for being energy efficient, the two protocols proposed in this paper implicitly achieve the similar goal. We have witnessed that for instance backoff with penalty reduces the total number of collisions from 30% to 14%. As a result the overall number of lost packets is also reduced (bear in mind that in 802.11 there are 7 retries before the frame is discarded). Although we do not have a quantitative estimation of the amount of saved energy, and have not explored the relationship between collisions and energy consumption, we believe that the proposed protocols have a good potential for energy saving.

*Can we deal with hidden terminals?* Hidden terminals is a well understood problem in wireless networks. And as we have demonstrated in Section IV-C, standard 802.11 backoff protocol lacks fairness in such settings (at least when the number of stations is 2). Potentially, such mechanism as RTS/CTS allows to avoid the problem but at the cost of extra overhead.

Our goal, on the other hand, was to understand whether penalty and reward mechanisms are sufficient to tackle the hidden terminal problem. We have found that even though for  $r \leq 1.5$  backoff with penalty shows little fairness, already for  $r > 1.5$  the protocol passes the threshold level of 0.7, which can be considered as good fairness, good enough to give all nodes possibility to transmit packets.

Poor fairness of the protocol with penalty when  $r$  is small

is the result of almost equal values of expected backoff time and time needed to successfully send a packet. We have observed that when  $r = 1.4$  the expected backoff time for the protocol on average will be  $t_b = 540\mu s$ . At the same time, our calculations suggest that on average it takes  $t_s = 510\mu s$  to successfully transmit a packet at the rate of  $36Mb/s$  (this is the average rate observed in the experiment with two hidden stations). This means that it is very likely that one of the stations will start sending a packet while the other one still hasn't finished doing so. Of the two colliding packets the access point will receive the one sent by the station with higher transmitting power, resulting in the other station consistently having a smaller fraction of successfully transmitted packets. Ideally, for the scheme to work time needed to transmit a packet should be considerably smaller than the expected backoff time. Indeed, we have witnessed that backoff with penalty achieves good fairness already when  $r = 1.6$  (which can be selected as minimum for real deployments as nodes will not lose much in terms of throughput), *i.e.* when  $t_b > 1ms$ .

Potentially, as  $N$  gets larger the backoff with penalty should naturally resolve the problem. According to theory, already when  $N > 3$ ,  $r$  gets larger than 1.4 which implies  $t_b \sim 0.8ms$ . Throughout experimentation we have seen that in such configuration the protocol shows adequate performance in terms of fairness.

*Adapting the backoff factor and estimation of active stations?* In standard 802.11 backoff protocol throughput grows with the increase of  $r$  (which is however severely penalized with reduced fairness) and the trend holds for all  $N$  we have experimented with. Though as our results suggest, relation between throughput and  $r$  is different for backoff with penalty protocol, where an optimal value of  $r$  has to be selected for each  $N$ . In Table III we refer to this property as *N-adaptable*. Indeed, setting  $r$  too large when  $N$  is small will result in a large number of wasted slots; choosing a too small  $r$  when  $N$  is large – in large number of collisions. In Section V we have demonstrated that  $r$  should be a function of  $N$ .

In practice, however, counting the number of active stations in a distributed way can be a challenging task. For instance, in Idle Sense access protocol [12] the authors suggested to count the number of idle slots between any two transmissions. And it is these idle slots that are used as an indication to increase or decrease the window size. Unfortunately, such approach will not work in some environments, *e.g.* when hidden terminals are present – nodes will simply miscount "occupied" slots. As an alternative, the nodes can use the information provided or relayed by an access point. In this work, we demonstrate how access points can estimate the number of active stations and broadcast this information to attached nodes. In a mesh network deployment there has to be some other distributed algorithm to find the number of active neighbors in the vicinity of a particular node. We believe that this issue deserves a separate investigation and leave it out for future work.

*Deployment.* We foresee at least two possible deployment paths. First, our protocol can be deployed on the nodes and coexist in a way similar to 802.11b and 802.11g protocols: If all associated stations support modified version of backoff protocol, then they all use it. Otherwise, if at least one legacy

host enters the network, all nodes fallback to unmodified backoff protocol. Second, the protocols we discuss can be readily deployed in such networks where all nodes are guaranteed to follow it. For instance, in mesh networks backbone nodes can use modified backoff protocol to communicate between each other, and use unmodified backoff for communication with legacy clients (obviously using a different wireless channel).

*A tussle with greedy and unfair: can penalty mechanisms coexist with greedy protocols such as standard 802.11?* An interesting situation arises when, for instance, the backoff with penalty is deployed in the environment where greedy protocols prevail. Intuitively, the more greedy standard backoff can easily capture the channel, because it doesn't feature self-punishment for successful transmissions. On the other hand, the nodes that follow backoff with penalty will remain fair between each other, but will inadequately self-penalize and as a result will have smaller throughput.

We think that the study of this tussle problem deserves a closer look and can be formulated separately. But to outline, we consider it possible to find a protocol from the family of penalty-based backoffs which potentially can tackle the problem. We leave this for future work.

## VIII. RELATED WORK

The literature on wireless communication is so extensive that in the proceeding paragraphs we can only summarize a few of the many lines of research related to what we discuss in this paper.

*Modeling and Simulation.* This body of work is arguably the largest and mainly focuses on modeling the performance of standard 802.11 protocol [3], [5], [7], [8], [17], [20], [23], [24], [27], [29]. Although some of the techniques described in this literature are similar to ours, we unlike other studies, confirm our hypothesis with real-life experiments.

*Optimizations.* The literature in this area is rife with various approaches and here we outline just two major research directions. The first one is focused on modifications of backoff protocol [12], [18], [25], [26]. Unlike these works, which suggest to either remove the exponential backoff completely and use fixed contention windows or use non-standard backoff factors, we take a radical step and propose to use non-standard state transitions to penalize certain stations. The other body of work focuses on the application of efficient coding schemes [9], [11], [14]–[16]. This does not bear directly on our approach, but it does illustrate one way to cope with collisions in wireless networks. Moreover, we consider these approaches to be supplementary to our work.

*Experimental work.* We find it surprising that there exists little empirical investigation of backoff protocols. As such, we have found that only in [10] the authors reported some practical implementation and evaluation of backoff protocol using proprietary hardware and firmware and a small number of wireless stations. And few research works [28] consider implementation of MAC protocols in general on commodity hardware. Thus, our work is another step towards better understanding of real-life performance of modified MAC protocols on commodity hardware.

## IX. CONCLUSION

The main goal of this paper is to evaluate performance of various 802.11 backoff protocols in real-life experiments. The four protocols we considered were standard backoff, backoff with fixed contention window, backoff with penalty and rollback backoff. The first step in our work was to implement the protocols on suitable devices and design the experimental setup. We described in details the difficulties we had encountered during implementation of backoff protocols on commodity wireless cards, since we believe that these details may help other researchers who will be experimenting with the same or similar devices.

We put considerable effort in designing our experiments and making sure that our data is consistent. Our study includes several scenarios: in lossy and normal environment, with upload and download traffic pattern, with hidden stations and delay sensitive applications, with TCP and UDP flows. In the experiments we observed the consequences of varying backoff factor from 1.2 to 2.6 and number of stations from 3 to 12. The collected data was carefully scrutinized and calibrated to avoid any errors.

The three main performance metrics we have monitored were aggregated throughput, short-term fairness and collision probability. In our experiments standard 802.11 backoff protocol and backoff with fixed contention window demonstrated poor fairness. We have concluded that unfair behavior is built-in to the standard 802.11 backoff protocol, where unsuccessful stations are forced to remain unsuccessful, while successful stations are able capture the shared medium for long periods and remain successful.

To mitigate the problem with fairness, we have introduced self-penalty mechanism in backoff with penalty and rollback backoff protocols. Our empirical evidence reveals that these two protocols achieve nearly perfect fairness and throughput up to 100% higher than in standard backoff. In addition, the penalty mechanism allowed to decrease collision rate by more than half and to reduce delays in delay sensitive traffic.

## REFERENCES

- [1] <http://www.ing.unibs.it/~openfwwf/>.
- [2] K. Barr and K. Asanović. Energy aware lossless data compression. In *ACM MobiSys '03*, pages 231–244, 2003.
- [3] G. Bianchi. Performance analysis of the IEEE 802.11 distributed coordination function. *IEEE Journal on Selected Areas in Communications*, 18(3):535–547, Mar 2000.
- [4] Y.-C. Cheng, J. Bellardo, P. Benkö, A. C. Snoeren, G. M. Voelker, and S. Savage. Jigsaw: solving the puzzle of enterprise 802.11 analysis. In *ACM SIGCOMM '06*, pages 39–50, 2006.
- [5] J.-W. Cho and Y. Jiang. Fundamentals of the Backoff Process in 802.11. *CoRR*, abs/0904.4155, 2009.
- [6] J. Deng, P. K. Varshney, and Z. Haas. A new backoff algorithm for the IEEE 802.11 Distributed Coordination Function. In *CNDS '04*, pages 215–225.
- [7] A. Duda. Understanding the performance of 802.11 networks. In *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 1–6, 2008.
- [8] E. Felemban and E. Ekici. Single Hop IEEE 802.11 DCF Analysis Revisited: Accurate Modeling of Channel Access Delay and Throughput for Saturated and Unsaturated Traffic Cases. *IEEE Transactions on Wireless Communications*, 10(10):3256–3266, 2011.
- [9] S. Gollakota and D. Katabi. Zigzag decoding: combating hidden terminals in wireless networks. In *SIGCOMM '08*, pages 159–170, 2008.
- [10] Y. Gruenberger, M. Heusse, F. Rousseau, and A. Duda. Experience with an implementation of the idle sense wireless access method. In *ACM CoNEXT '07*, pages 24:1–24:12.
- [11] A. Gudipati and S. Katti. Strider: automatic rate adaptation and collision handling. In *ACM SIGCOMM '11*, pages 158–169.
- [12] M. Heusse, F. Rousseau, R. Guillier, and A. Duda. Idle sense: An optimal access method for high throughput and fairness in rate diverse wireless LANs. In *ACM SIGCOMM*, pages 121–132. ACM Press, 2005.
- [13] R. Jain. *The Art of Computer Systems Performance Analysis*. 1991.
- [14] S. Katti, S. Gollakota, and D. Katabi. Embracing wireless interference: analog network coding. In *ACM SIGCOMM '07*, pages 397–408, 2007.
- [15] S. Katti, D. Katabi, H. Balakrishnan, and M. Medard. Symbol-level network coding for wireless mesh networks. In *ACM SIGCOMM '08*, pages 401–412, 2008.
- [16] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft. Xors in the air: practical wireless network coding. In *ACM SIGCOMM '06*, pages 243–254, 2006.
- [17] A. Kumar, E. Altman, D. Miorandi, and M. Goyal. New insights from a fixed-point analysis of single cell IEEE 802.11 WLANs. *IEEE/ACM Trans. Netw.*, 15(3):588–601, 2007.
- [18] B.-J. Kwak, N.-O. Song, and L. E. Miller. Performance analysis of exponential backoff. *IEEE/ACM Trans. Netw.*, 13(2):343–355, 2005.
- [19] M. Lacage, M. H. Manshaei, and T. Turletti. IEEE 802.11 rate adaptation: a practical approach. In *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, MSWiM '04, pages 126–134, New York, NY, USA, 2004. ACM.
- [20] A. Lukyanenko and A. Gurtov. Performance analysis of general backoff protocols. *Journal of Communications Software and Systems*, 4(1), March 2008.
- [21] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, New York, NY, USA, 2007.
- [22] S. Sharafeddine and R. Maddah. A lightweight adaptive compression scheme for energy-efficient mobile-to-mobile file sharing applications. *Journal of Network and Computer Applications*, 34:52–61, January 2011.
- [23] N.-O. Song, B.-J. Kwak, J. Song, and L. E. Miller. Enhancement of IEEE 802.11 distributed coordination function with exponential increase exponential decrease backoff algorithm. *IEEE VTC*, 2003:2775–2778, 2003.
- [24] N.-O. Song, B.-J. Kwak, J. Song, and L. E. Miller. Analysis of EIED backoff algorithm for the IEEE 802.11 DCF. In *Proc. of Vehicular Technology Conference, VTC2005-Fall*, volume 4, pages 2182–2186. IEEE, 2005.
- [25] P. Starzetz, M. Heusse, F. Rousseau, and A. Duda. Hashing backoff: A collision-free wireless access method. In *IFIP-TC 6 NETWORKING '09*, pages 429–441.
- [26] K. Tan, J. Fang, Y. Zhang, S. Chen, L. Shi, J. Zhang, and Y. Zhang. Fine-grained channel access in wireless lan. In *ACM SIGCOMM '10*, pages 147–158, 2010.
- [27] Y. C. Tay and K. C. Chua. A capacity analysis for the IEEE 802.11 MAC protocol. *Wireless Networks*, 7:159–171, March 2001.
- [28] I. Tinnirello, G. Bianchi, P. Gallo, D. Garlisi, F. Giuliano, and F. Gringoli. Wireless MAC processors: Programming MAC protocols on commodity hardware. In *INFOCOM*, pages 1269–1277, 2012.
- [29] H. Vu and T. Sakurai. Collision Probability in Saturated IEEE 802.11 Networks. In *Australian Telecommunication Networks and Application Conference 2006*, pages 21–25.
- [30] Y. Xiao, P. Savolainen, A. Karppanen, M. Siekkinen, and A. Ylä-Jääski. Practical power modeling of data transmission over 802.11g for wireless applications. In *e-Energy '10*, pages 75–84.
- [31] J. Zhang, K. Tan, J. Zhao, H. Wu, and Y. Zhang. A practical SNR-guided rate adaptation. In *INFOCOM '08*, pages 2083–2091, 2008.